IBM PROJECT-MALWARE DETECTION

GROUP 10

Nithin Krishna

AIM

Detection and Prevention of Advanced Persistent Threat (APT) activities in heterogeneous networks using SIEM and Deep Learning.

FEATURES

- Our observation also includes a multi class classified output with various types of attacks like DoS, Probe, U2R, etc…

- Hence, using these common features, we understand that when a new data point is provided and asked to classify under a type of attack with these respective columns' data, we can find its type of attack.

1. Protocol
2. Service
3. Flag
4. Duration
5. level

DATASET DESCRIPTION

The dataset I've have considered is NSL-KDD

The feature types in this data set can be broken down into 4 types:

4 Categorical (Features: 2, 3, 4, 42)

6 Binary (Features: 7, 12, 14, 20, 21, 22)

23 Discrete (Features: 8, 9, 15, 23-41, 43)

10 Continuous (Features: 1, 5, 6, 10, 11, 13, 16, 17, 18, 19)

Flag	Value	Flag	Description
SF	Normal establishment and termination. Note that this is the same symbol as for state S1. You can tell the two apart because for S1 there will not be any byte counts in the summary, while for SF there will be	RSTO	Connection reset by the originator
REJ	Connection attempt rejected	RSTR	Connection reset by the responder
S0	Connection attempt seen, no reply	OTH	No SYN seen, just midstream traffic (a “partial connection” that was not later closed)
S1	Connection established, not terminated	RSTOS0	Originator sent a SYN followed by a RST, we never saw a SYN-ACK from the responder
S2	Connection established and close attempt by originator seen (but no reply from responder)	SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection was “half” open)
S3	Connection established and close attempt by responder seen (but no reply from originator)	SHR	Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator. (Not in NSL-KDD but still a flag)

Protocol Type (2)	Service (3)				Flag (4)
• icmp • tcp • udp	• other • link • netbios_ssn • smtp • netstat • ctf • ntp_u • harvest • efs • klogin • systat • exec • nntp • pop_3 • printer • vmnet • netbios_ns	• urh_i • ssh • http_8001 • iso_tsap • aol • sql_net • shell • supdup • auth • whois • discard • sunrpc • urp_i • Rje • ftp • daytime • domain_u • pm_dump	• time • hostnames • name • ecr_i • bgp • telnet • domain • ftp_data • nnsp • courier • finger • uucp_path • X11 • imap4 • mtp • login • tftp_u • kshell	• private • http_2784 • echo • http • ldap • tim_i • netbios_dgm • uucp • eco_i • Remote_job • IRC • http_443 • red_i • Z39_50 • Pop_2 • gopher • Csnet_ns	• OTH • S1 • S2 • RSTO • RSTRs • RSTOS0 • SF • SH • REJ • S0 • S3

Dataset	Number of Records:					
	Total	Normal	DoS	Probe	U2R	R2L
KDDTrain+20%	25192	13449 (53%)	9234 (37%)	2289 (9.16%)	11 (0.04%)	209 (0.8%)
KDDTrain+	125973	67343 (53%)	45927 (37%)	11656 (9.11%)	52 (0.04%)	995 (0.85%)
KDDTest+	22544	9711 (43%)	7458 (33%)	2421 (11%)	200 (0.9%)	2654 (12.1%)

Classes:	DoS	Probe	U2R	R2L
Sub-Classes:	• apache2 • back • land • neptune • mailbomb • pod • processtable • smurf • teardrop • udpstorm • worm	• ipsweep • mscan • nmap • portsweep • saint • satan	• buffer_overflow • loadmodule • perl • ps • rootkit • sqlattack • xterm	• ftp_write • guess_passwd • httptunnel • imap • multihop • named • phf • sendmail • Snmpgetattack • spy • snmpguess • warezclient • warezmaster • xlock • xsnoop
Total:	11	6	7	15

ALGORITHM USED AND APPROACH

- We have used many algorithms on our dataset such as K Nearest Neighbour (KNN) , Decision Tree Classifier , XG Boost & Artificial Neural Network (ANN) on the dataset to obtain the confusion matrix and a classification report & an Apriori algorithm that searches for a series of frequent sets of items in the datasets.

- To begin, we have performed :

    1. Calculated variance — columns with 0 variance have been dropped
    2. Plotted a heat map to check for independent features that are highly correlated to each other and dropped them.
    3. Converted all the attacks into distinct categories(DOS,Probe,U2R,R2l)
    4. Calculated the impact of the independent features on Y and dropped the ones which do not constitute much of an effect.
    5. Applied one hot encoding on the categorical variables to convert strings to numbers.
    6. Split dataset into Test and Train datasets
    7. Scaling - Used StandardScaler() - to normalise all values
    8. Applied KNN ,Decision Tree Classifier, XG Boost , ANN & Apriori algorithms on the dataset and found the results.

PERFORMANCE MATRIX

1. Accuracy Score = 0.8187907554451492
   Algorithm Used : **KNN**

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[9429   24  245    5    8]
 [ 873 6411   41  133    1]
 [ 314  208 1893    6    0]
 [ 943  731  459  705   47]
 [   5    1   16   25   20]]
0.8187907554451492
```

2. Accuracy Score = 0.8979727631637315
   Algorithm Used : **Decision Tree Classifier**

```
[136] from sklearn.metrics import confusion_matrix, accuracy_score
      cm = confusion_matrix(y_test, y_pred)
      print(cm)
      accuracy_score(y_test, y_pred)
```

```
[[9711    0    0    0    0]
 [   0 6536  347  574    2]
 [   0  198 2026  195    2]
 [   0  444  399 1943   99]
 [   0    0    2   37   28]]
0.8980171228319213
```

3. Accuracy Score = 0.8979727631637315
   Cross Validation Score: 99.89 %
   Algorithm Used : **XG Boost**

```
[ ] from sklearn.metrics import confusion_matrix, accuracy_score
    y_pred = classifier.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    accuracy_score(y_test, y_pred)
```

```
[[9711    0    0    0    0]
 [   0 7150  193  114    2]
 [   0  350 1996   75    0]
 [   0  956  562 1348   19]
 [   0    0    4   36   27]]
0.8974848068136451
```

```
[ ] from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(estimator = classifier, X = x_train, y = y_train, cv = 10)
    print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
    print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 99.89 %
Standard Deviation: 0.03 %
```

4. Algorithm Used : **ANN**
   Accuracy Score = 0.990

```
Epoch 96/100
3936/3936 [==============================] - 7s 2ms/step - loss: 0.0051 - accuracy: 0.9991
Epoch 97/100
3936/3936 [==============================] - 7s 2ms/step - loss: 0.0052 - accuracy: 0.9991
Epoch 98/100
3936/3936 [==============================] - 7s 2ms/step - loss: 0.0053 - accuracy: 0.9991
Epoch 99/100
3936/3936 [==============================] - 8s 2ms/step - loss: 0.0051 - accuracy: 0.9990
Epoch 100/100
3936/3936 [==============================] - 7s 2ms/step - loss: 0.0053 - accuracy: 0.9990
<keras.callbacks.History at 0x7f7025aefd90>
```

Apriori —

Over relational databases, Apriori is an algorithm for frequent item set mining and association rule learning. It works by recognising the most common individual items in the database and expanding them to bigger and larger item sets as long as those item sets exist in the database frequently enough. We can see the results we have obtained in the following picture below :

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	RSTR	probe	0.014917	0.728032	8.257248
1	Z39_50	S0	0.004710	0.760221	3.024073
2	courier	S0	0.004135	0.783398	3.116270
3	iso_tsap	S0	0.003888	0.772789	3.074068
4	nnsp	S0	0.003478	0.758209	3.016070
5	supdup	S0	0.003094	0.791594	3.148871
6	vmnet	S0	0.003409	0.756839	3.010620
7	whois	S0	0.003841	0.768493	3.056979
8	domain_u	udp	0.067638	1.000000	8.337443
9	eco_i	icmp	0.032470	1.000000	16.523982
10	eco_i	probe	0.028917	0.890576	10.100796
11	ecr_i	icmp	0.023687	1.000000	16.523982
12	ftp	r2l	0.006243	0.373006	14.330813
13	r2l	ftp_data	0.007209	0.276959	5.251302
14	icmp	probe	0.029225	0.482919	5.477202
15	urp_i	icmp	0.004279	1.000000	16.523982
16	other	probe	0.011597	0.370273	4.199592
17	other	udp	0.018217	0.581639	4.849385
18	pop_3	r2l	0.004895	0.573376	22.028983

REFERENCES:

https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657

LINK TO DATASET:

https://www.unb.ca/cic/datasets/nsl.htm

GITHUB REPOSITORY :

https://github.com/nithin0905/malware_analysis